Using Python to Program LEGO MINDSTORMS® Robots: The PyNXC Project

Brian S. Blais
Bryant University, Smithfield RI
bblais@bryant.edu

Abstract

LEGO MINDSTORMS® NXT (Lego Group, 2006) is a perfect platform for introducing programming concepts, and is generally targeted toward children from age 8-14. The language which ships with the MINDSTORMS®, called NXTg, is a graphical language based on LabVIEW (Jeff Kodosky, 2010). Although there is much value in graphical languages, such as LabVIEW, a text-based alternative can be targeted at an older audiences and serve as part of a more general introduction to modern computing. Other languages, such as NXC (Not Exactly C) (Hansen, 2010) and PbLua (Hempel, 2010), fit this description. Here we introduce PyNXC, a subset of the Python language which can be used to program the NXT MINDSTORMS®. We present results using PyNXC, comparisons with other languages, and some challenges and future possible extensions.

1. Introduction

LEGO MINDSTORMS® is a robotics platform where the robot structure is built with LEGOs® and the programs are entered on a computer and downloaded via a USB connection. The robot is then disconnected and run autonomously. The choice of languages fall into two categories: graphical and text-based. Because the product is marketed primarily towards the elementary and middle school, through such events as the FIRST LEGO League and Robotics Park(Kelly and Daudelin, 2008), the graphical languages are the ones that are officially supported by the LEGO company. These include RobotLab and NXT-G. There are third-party text-based languages based on well-known programming languages like C, Java, and Lua. These include NXC, Robot C, LeJOS, PbLua, and others.(Hassenplug, Steve, 2008) NXC is the most widely used unofficial language. The only solutions for programming robots with Python either use the bluetooth interface (pynxt, 2010), where the Python interpreter is not running on the robot but on an external computer, or don't support LEGO MINDSTORMS® (Blank et. al, 2006; myro, 2010).

When starting the PyNXC project (Blais, 2010), my goal was to find a fully autonomous solution or the LEGO MINDSTORMS® using Python. However, the robot's firmware places limits on any language that one uses, especially limits on memory size and dynamic allocation, such as limiting the dynamic allocation to arrays. These limitations can be overcome with a custom firmware, like the Java solution LeJOS, but PyNXC aims at using the firmware which comes with the NXT, so that it can work "out of the box". Instead of writing NXT bytecode directly, the

PyNXC project, as the name suggests, translates Python code into NXC code and then uses the NXC compiler to download the program to the robot.

1.1. NXC

NXC (Not Exactly C) is the most widely used third-party, free, library for programming the NXT Robots. It uses C-like syntax, but is not a full C implementation due to the limitations of the robot firmware.

A simple program, out of the NXC Tutorial (Benedettelli, 2007) is the following program which moves forward until the touch-sensor (on NXT port 1) is pressed:

```
#include "NXCDefs.h"
task main()
{
  SetSensor(IN_1,SENSOR_TOUCH);
  OnFwd(OUT_AC, 75);
  until (SENSOR_1 == 1);
  Off(OUT_AC);
}
```

The sensor is defined to be a touch sensor (which will return a 1/0 value for pressed/not pressed), and the motors on ports A and C are set to 75% power. The robot rolls forward until the sensor on port 1 is triggered. At this point the motors are turned off. For those used to Python, the extra syntax (braces, semicolons, lack of indented code blocks, etc...) is seen as visual noise, and obstructs the later understanding of the code.

1.2. PyNXC

A python-to-NXC converter is an ill-defined concept. NXC is statically typed while python is dynamically typed. Python is much more general than NXC, and does not have the same data structures. NXC has structs, arrays, and mutices while python has classes, lists, and a huge standard library. At first it would seem that this project would have the same goals as other Python-to-C converters, such as Cython (Cython, 2010) and Shedskin (Dufour, 2010), but the robot hardware changes the goals. In Shedskin, for example, the conversion attempts to infer the data type, such as int or float. When programming robots, the memory requirements are such that one is always concerned with the data size, so the programmer chooses byte, short, or int as they need to for optimization. These types cannot, even in principle, be inferred. Cython's goal includes "compiling regular Python code and (most of) the normal Python test suite" (Cython, 2010). Optimizations needed for the robots may require some non-python syntax to be included in the PyNXC project, and will certainly severely limit the scope of the python code supported.

The reason I am writing this project is so that I can use Python *syntax* to program robots, so that in my courses (where there are both robot and non-robot programming assignments) I can use one language. Pedagogically speaking, this is a critical goal especially for beginning programmers where even slight differences in the syntax of two languages causes problems.

2. Example Programs

In this section I include a number of simple examples, to give the reader a flavor of PyNXC. I examine some of the non-pythonic syntax included, for optimization, and look at some limitations and future directions.

2.1. Examples

The first example is a translation of the NXC code above.

```
def main():
    DefineSensors(TOUCH,None,None,None)
    OnFwd(OUT_AC, 75)
    while  SensorVal(1) != 1:
        pass
    Off(OUT_AC)
```

Here, the difference in the program is not substantial, but the program is a bit cleaner than the NXC version. There are no "until" statements in Python, so a while-loop is used. There are no `#include` preprocessor instructions in Python, so the boilerplate code is written into the translator.

A more complex example is the line-finding task shown in Figure 1. What makes this task more complex is that the robot has to determine, on its own, the distance to the wall (using the included ultrasonic range sensor), the location of the barrier (either right or left), and the location of the line on the ground (using the light sensor included in the NXT kit).

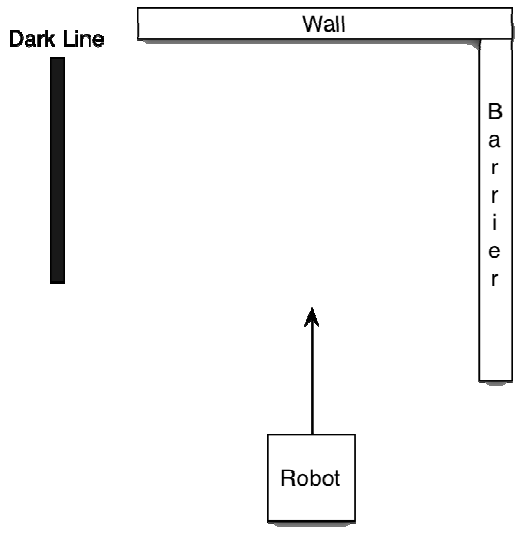

Figure 1. Line-finding task for the Robot. The robot needs to go forward to the wall, turn away from the barrier, move to the dark line on the ground, and stop. The robot does not know on which side the barrier will be, the right or the left. Shown is the barrier on the right with the dark line on the left. The reverse configuration is also possible, with the barrier on the left and the line on the right.

The code for solving the line-finding task is the following:

```
distance=0
right_wall=0
stop_check=0

def Turn(ang):
    if ang>0: # CW
        OnFwd(OUT_C,80)
        OnRev(OUT_A,80)
        Wait(230*abs(ang)/90)  # with power 80
    else:     # CCW
        OnFwd(OUT_A,80)
        OnRev(OUT_C,80)
        Wait(280*abs(ang)/90)  # with power 80

    Off(OUT_AC)

def GoToLine():
    OnFwd(OUT_A,100)
    OnFwd(OUT_C,100)

    # go to dark line
    while SensorVal(2)>30:
        pass

    Off(OUT_AC)

def task_CheckRight():
    RotateMotor(OUT_B,100,90)  # turn the eyes to look right

    while not stop_check:
        if SensorVal(4)<50:  # object is close on the right
            right_wall=1     # ...so it must be a barrier
            stop_check=1

    RotateMotor(OUT_B,-100,90)

def task_Forward():
    RotateMotor(OUT_AC,100,distance*15)
    stop_check=1

    if right_wall:
        Turn(-90)
    else:
        Turn(90)

    GoToLine()

def main():

    DefineSensors(SOUND,LIGHT,None,EYES)

    # wait for a clap
    while SensorVal(1)<50:
        pass

    distance=SensorVal(4)  # distance to the wall, in cm
    StartTask(task_Forward)
    StartTask(task_CheckRight)
```

I will explain the general flow of the program here, and along the way highlight some of the aspects of the Python to NXC translation. In every program there is a function called `main` which gets called when the robot is told to run the program..

The NXT is a multitasking platform, where all of the initial tasks are started in the main program and are run concurrently. In NXC, tasks are denoted with a task keyword. In PyNXC,

```
def task_foo():
    pass
```

is translated to NXC code:

```
task task_foo(){
}
```

In this way, the name of a function can determine what kind of NXC function is involved. This convention works for functions declared as inline (e.g named `inline_foo`) and subroutines (e.g. named `sub_foo`).

The two tasks are started concurrently, and use global variables `distance`, `right_wall`, and `stop_check` to communicate information. There are mutex variables in NXC which can be used to avoid collisions when tasks need to write information to the same place. Variables that are not declared as a certain type as assumed to be of time `int`. In this way, some of the dynamism of Python is retained, and the boilerplate code of type declarations is largely avoided in simple programs.
The task `task_Forward` rolls forward, sets the `stop_check` flag (to signal to the other task to end, and then turns right or left depending on the value of the `right_wall` flag set by the other task. Functions work just as in Python, and all of the API calls of NXC are available.

Many more examples can be found on the project website, http://code.google.com/p/pynxc/, including a translation of all of the programs in the NXC Tutorial.

2.2. Other Capabilities

The following is a miscellaneous list of the way PyNXC translates to particular NXC syntax elements.

- A `DEFINE` keyword is introduced to do preprocessor search-and-replace. It is equivalent to the `#DEFINE` in NXC.

```
DEFINE turn_around=OnRev(OUT_B, 75); Wait(3400); OnFwd(OUT_AB, 75);

def main():
    OnFwd(OUT_AB, 75)
    Wait(1000)
    turn_around
    Wait(2000)
    Off(OUT_AB)
```

- NXC `structs` are implemented as Python classes, subclassing `Struct`.

```
class MyStruct(Struct):
    x=Byte(5)
    y=Word(6)
    s=String('hello')

m=MyStruct()
```

```
m2=MyStruct(x=6,y=7,s='this')
```

- `Typedefs` are implemented just like `struct`, by the Python subclassing syntax.

```
class MyByte(Byte):
    pass
```

- `For`-loops work with limited forms `range`, not on lists (which don't exist in NXC) or arrays (yet).

```
i=Byte()
s=String()

for i in range(256):
    s=NumToStr(i)

for i in range(1,12):
    s=NumToStr(i)

for i in range(1,12,3):
    s=NumToStr(i)
```

2.3. Limitations

Recent versions of NXC (version b36) include floating-point support, which PyNXC does not currently support. Some new functions on arrays are also not supported yet, but should be shortly. Clearly this project has all of the limitations that NXC has, but at the same time it also automatically improves when more API calls are included in NXC.

3. Conclusions

Ideally, it would be nice to have a Python virtual machine working the NXT platform, like LeJOS does with the Java virtual machine. This would automatically lead to a huge flexibility increase in the use of Python for the NXT. However, given the memory limitations of the NXT, there would probably be some necessary sacrifice of the power of Python to be practical. The PyNXC project fills the middle-ground, where one gets at least some of the benefits of programming in Python on the NXT platform, with a very small memory footprint of NXC. It works "out of the box" with the firmware which ships with the NXT, and can be used as a spring-board for novice programmers to learn python and apply it to more advanced problems.

4. References

Benedettelli, Daniele (2007). "Programming LEGO NXT Robots using NXC". [Online] Available at http://bricxcc.sourceforge.net/nbc/nxcdoc/NXC_tutorial.pdf. Retrieved 2010-02-26.

Blais, Brian (2010). "pynxc: A Python to NXC Converter for Programming LEGO Mindstorms Robots". [Online] Available at http://code.google.com/p/pynxc/. Retrieved 2010-02-26.

Blank, D.S., Kumar, D., Meeden, L., and Yanco, H. (2006) "The Pyro toolkit for AI and robotics". In *AI Magazine* Volume 27, Number 1.

Cython (2010). "C-extensions for Python". [Online] Available at http://www.cython.org/. Retrieved 2010-02-26.

Dufour, Mark (2010). "shedskin: An experimental (restricted) Python-to-C++ compiler". [Online] Available at http://code.google.com/p/shedskin/. Retrieved 2010-02-26.

Hansen, John (2010). "Welcome to Next Byte Codes and Not eXactly C". [Online] Available at http://bricxcc.sourceforge.net/nbc/. Retrieved 2010-02-26.

Hassenplug, Steve (2008). "NXT Programming Software" [Online] Available at http://www.teamhassenplug.org/NXT/NXTSoftware.html. Retrieved 2010-02-26.

Hempel, Ralph (2010). "PbLua Home Page" [Online] Available at http://www.hempeldesigngroup.com/lego/pbLua/index.html Retrieved 2010-02-26.
Kelly, James and Daudelin, Jonathan (2008). "First LEGO League: The Unofficial Guide". No Starch Press.

Kodosky, Jeff (2010). "Is LabVIEW a general purpose programming language?". [Online] Available at http://zone.ni.com/devzone/cda/tut/p/id/5313. Retrieved 2010-02-26.
Lego Group (January 4, 2006). "What's NXT? LEGO Group Unveils LEGO MINDSTORMS NXT Robotics Toolset at Consumer Electronics Show". Press release. Retrieved 2010-02-26.

Myro Development. [Online] Available at http://wiki.roboteducation.org/Myro_Development. Retrieved 2010-02-26.

PyNXT Project. [Online] Available at http://pynxt.sourceforge.net/ Retrieved 2010-02-26.